PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
H04L 12/24

A1

(11) International Publication Number: WO 99/34557

(43) International Publication Date: 8 July 1999 (08.07.99)

(21) International Application Number: PCT/US98/27390

(22) International Filing Date: 22 December 1998 (22.12.98)

(30) Priority Data:
08/998,169 24 December 1997 (24.12.97) US

(71) Applicant: QUALCOMM INCORPORATED [US/US]; 6455 Lusk Boulevard, San Diego, CA 92121 (US).

(72) Inventors: GEHLHAAR, Jeff, B.; 11934 Dapple Way, San Diego, CA 92128 (US). DOLTER, James, W.; 11755 Timberlake Drive, San Diego, CA 92131 (US). RAM, Siddhart, R.; 7920 Avenida Navidad #147, San Diego, CA 92122 (US). ANAND, Rahul; 927 Wilbur Avenue #3, San Diego, CA 92109 (US).

(74) Agents: MILLER, Russell, B. et al.; Qualcomm Incorporated, 6455 Lusk Boulevard, San Diego, CA 92121 (US).

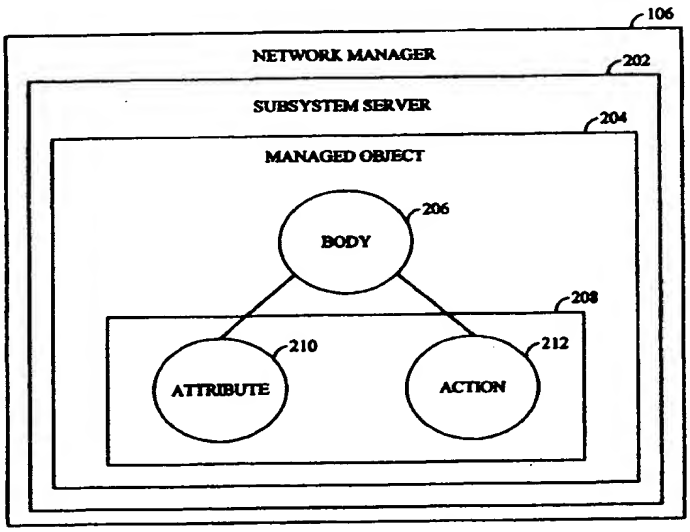
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND SYSTEM FOR SOFTWARE VERSION MANAGEMENT IN A NETWORK MANAGEMENT SYSTEM



(57) Abstract

A system and method for software version management, including a network management system (100), network manager (106), and at least one network element is described. The software version management process identifies a first managed object tree (600) having a plurality of managed objects to be upgraded. The managed object tree (600) is used to generated a second managed object tree. The second managed object tree is generated by transforming the topology of the first managed object tree. After the second managed object tree has been generated, the plurality of managed objects (602) are data transformed. Data transformation is accomplished by data transformation functions executed to upgrade the individual managed objects.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND SYSTEM FOR SOFTWARE VERSION MANAGEMENT IN A NETWORK MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

I. Field of the Invention

The present invention relates generally to network management systems, and more specifically is directed toward software version management.

II. Related Art

Telecommunication service providers provide a wide range of services to their customers. These services range from the transport of a standard 64 kbit/s voice channel (i.e., DS0 channel) or subrate thereof to the transport of higher rate digital data services (e.g., video). Both voice channels and digital data services are transported over the network via a hierarchy of digital signal transport levels. For example, in a conventional digital signal hierarchy 24 DS0 channels are mapped into a DS1 channel. In turn, 28 DS1 channels are mapped into a DS3 channel. The number of customers served and the complexity of services offered by telecommunication service providers are always increasing.

The wide range of services, signals and channels require a complex network of telecommunications equipment. Management of the complex telecommunications network is necessary to maintain optimum levels of service to the customer as well as efficiency in the maintenance and usage of the equipment itself. As networks grow increasingly complex, both in the size of the network and the range of services provided by the network, network management becomes increasingly important. Telecommunications service providers provide for management of the network by implementing network management systems designed to manage, provide for growth and ensure optimum performance of the network.

Network management systems include at least two layers. The first layer is the network manager layer. The network manager layer includes a network manager that monitors and controls the configuration of the network. The network manager is usually a server and software that maintains a logical representation of the state and condition of the network.

The network manager provides an interface to the network for users and applications wishing to manage the network.

The second layer of the network is the network element layer. The network element layer includes all of the systems and hardware not associated with the task of managing the network. Such systems are defined as network elements. Examples of network elements are the mobile switching center (MSC), call detail adjunct (CDA), home location registry (HLR), channel service unit (CSU), customized dial plan (CDP), CDMA interconnect subsystem (CIS), etc. The network elements provide the functionality and services of the entire network, independent of the network manager.

The network management system is the combination of the network manager in the network manager layer and the network elements in the network element layer. The network management system implements a set of procedures, software, equipment and operations designed to keep the network operating near maximum efficiency. The goals of network management include configuration management, fault location and repair management, security management, and performance management.

Configuration management deals with installing, initializing, loading, modifying and tracking the configuration parameters and software of network elements and their associated software. The network manager accomplishes configuration management by downloading configuration parameters and software to the network elements. The network manager also tracks the configuration of the network by retrieving data indicating the configuration of the network elements and their associated software.

Fault location and repair management predicts and diagnoses problems with the network and provides a methodology for replacing or rerouting the network around the affected network elements. The network manager accomplishes fault location and repair management by retrieving fault information from the network elements in the network element layer. For example, the network manager may retrieve the number of severely errored seconds (SES) or the frame error rate (FER) from a network element in order to locate faults and diagnose problems with the network. If, upon retrieval for fault information from the network element layer, the network manager determines that particular network elements are experiencing degraded performance or are inoperative, the network manager may reroute the network around the affected elements. The network manager accomplishes the rerouting function by downloading additional configuration information to the network elements in order to reconfigure the network.

Security management allows the network manager to restrict access to various resources in the network, thereby giving customers different levels of access to different network resources. The network manager accomplishes security management by retrieving the current security information from the network elements and analyzing it. If the access to resources in the network is to be changed, the network manager accomplishes the change by downloading additional or changed security information to the network elements thereby changing the levels of access the users have to the network resources.

Performance management provides statistical information about the network's operation. This allows the network manager to manage the resources of the network to ensure optimum performance. The network manager monitors the usage and traffic levels of the network element to ensure that the traffic on the network is distributed. Distribution of traffic among the network elements helps to ensure that the network does not experience performance degradation because a few network elements are carrying most of the communications load, while other network elements are carrying too little. The network manager accomplishes performance management by retrieving information pertaining to the traffic loading of the particular network elements. If the network manager determines that the performance of the network would be improved by redistributing the network traffic from one set of network elements to another, the network manager downloads additional configuration information to the network elements, thereby reconfiguring them.

The functions of configuration management, fault location and repair management, security management, and performance management are usually accomplished by software executing on the network manager and network elements. A telecommunications network may contain tens of thousands of network elements. A network manager, therefore, may be managing tens of thousands of elements. As the functionality of the network manager and network elements evolve, the software in the network management system must change accordingly. The process of managing the process of upgrading software on the telecommunications network management system is called software version management.

Conventional telecommunication software version management processes upgrade network elements first, and then migrate the new software from the network elements to the network manager. Usually, the network element is loaded with the new software, and the old software is kept in place. A command is sent to the network elements to switch the

software over from the old version to the new version. In conventional software version management systems, the network elements are required to store two versions of software while the network management system is brought up to the current software version. After the network element and network manager have been upgraded, the switch to the new software version is accomplished by sending a command to the network element and switching the network manager. Conventional software version management systems, therefore, require that the network element be capable of storing and switching between two software versions, adding expense. In large telecommunications networks, the additional expense is multiplied by hundreds of thousands of network elements.

What is needed, therefore, is a network management software version management system and method which is capable of upgrading software in a network management system in which the network elements are not capable of storing and switching between two versions of software. Such a system would greatly decrease the cost of implementing and managing the network. The software version management system should provide a universal software upgrade method that upgrades the network management system software with a minimum of channel downtime.

SUMMARY OF THE INVENTION

The present invention is directed to software version management in telecommunications network management systems. The invention allows software to be upgraded while network management resources are available, without bringing the network management system down for an upgrade. A method of software version management is described and claimed for a network management system having a network manager and at least one network element, wherein the network manager has a plurality of managed object trees. First, the software version management process identifies a first managed object tree having at least one managed object to be upgraded. The software version management process creates a second managed object tree having at least one managed object by transforming the topology of the managed object tree. Next, the software version management process performs data transformation of the managed object(s) of the second managed object tree. The software version management process traverses the first managed object tree and builds a list of the managed object(s) of the first managed object tree. The software version management process identifies a topology transformation rule associated with the managed

object(s) of the first managed object tree. The software version management process generates at least one managed object of the second managed object tree.

BRIEF DESCRIPTION OF THE DRAWINGS

The features, objects, and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference numbers indicate identical or functionally similar elements. Additionally, the left-most digit of a reference number identifies the drawing in which the reference number first appears.

FIG. 1 is a block diagram of an exemplary network management system;

FIG. 2 is a block diagram illustrating an exemplary network manager;

FIG. 3 is a diagram illustrating the format of a message according to the present invention;

FIG. 4 is a diagram illustrating the format of a message element according to the present invention;

FIG. 5 is a block diagram illustrating an exemplary network element;

FIG. 6 is a diagram illustrating an exemplary managed object tree;

FIG. 7 is a diagram illustrating an example of topology transformation;

FIG. 8 is a diagram illustrating an example of containment domain managed object trees and operational domain managed object trees;

FIG. 9 is a flowchart illustrating the process of software version management;

FIG. 10 is a flowchart further illustrating the process of topology transformation;

FIG. 11 is a flowchart further illustrating the process of topology transformation; and

FIG. 12 is a block diagram illustrating an exemplary computer for implementing the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the description that follows, the first portion describes the preferred environment of the present invention. The second portion describes the software version management system. Both of these features comprise elements of this invention.

Network Management System

FIG. 1 illustrates a network management system 100 which is an exemplary environment for the present invention. Network management system 100 preferably complies with the International Telecommunications Union (ITU) telecommunications management network (TMN) standard. The TMN standard defines a layered framework for a service provider to implement its own network management processes.

Network management system 100 includes two network management layers 102 and 104. Layer 102 is designated as the network management layer 102. Network management layer 102 comprises network manager 106. Network manager 106 is logically shown as a single entity. In implementation, network manager 106 can comprise one or more sites. For example, multiple service centers (not shown) can exist at different parts of the country (e.g., east coast and west coast). Network manager 106 can also be split among services and/or network elements. For example, in one embodiment, a first network manager is dedicated to satellite-based communications, and a second network manager is dedicated to cell-based communications. Generally, the logical entity identified as network manager 106 is a resource that is accessed by client applications such as users and systems wishing to engage network management functions. Client applications access network manager 106 by transmitting messages to it and receiving messages from it.

Network management system layer 104 is designated as the network element layer 104. Network element layer 104 is the physical layer that includes the various network elements (e.g., mobile switching center, call detail adjunct, home location registry, channel service unit, customized dial plan, CDMA inter-connect subsystem, etc.) used in the transport and routing of network traffic. Each network element 108a-108n in network element layer 104 is designated to receive and transmit configuration, fault location, security and performance information associated with management of the network. In particular, network elements 108a-108n are connected to network manager 106 in network management layer 102. The present invention is applicable to and contemplates any management information passed between client applications and the network manager 106, or network manager 106 and network elements 108a-108n.

Although the exemplary environment for the present invention describes only two layers, alternative embodiments of network management system 100 having a plurality of layers are contemplated by the

present invention. For example, network management systems with a plurality of network management layers (more than two) or network management systems arranged hierarchically are equally suitable to the implementation of the present invention. In the case of multiple network management system layers, network manager 106 would interact with network elements 108a-108n through a plurality of communications protocols. Each layer in the network management system would interact with the network manager using a different protocol. Alternatively, network manager 106 could interact with network elements 108a-108n through an intervening network management layer (i.e., network manager 106 would interact with a first network entity, which in turn would interact with a second network entity in a different layer).

In the preferred embodiment, network management system 100 is a message based system. Client applications, such as users, applications or systems, interact with network manager 106 by transmitting messages to it. The messages usually contain requests or commands and are usually packetized. Likewise, network manager 106 communicates with network elements 108a-108n by transmitting and receiving similar messages. Such a messaging system is said to be asynchronous and transaction based. Transaction based systems rely upon messages and responses thereto in order to accomplish the management of the network. This is contrasted with connection based systems in which connections are established between external entities and network manager 106 or between network manager 106 and network elements 108a-108n. In asynchronous message based systems, such as network management system 100, messages are transmitted without acknowledgment from the network system or network entity to which the message is transmitted. A network entity is any network manager, network element, user or system that originates network management messages in the network management system 100.

FIG. 2 further illustrates network manager 106. The preferred embodiment of network manager 106 is a server and software that maintains a logical representation of the state and condition of network management system 100. Network manager 106 monitors and controls the configuration of the network by interacting with network elements 108a-108n in network element layer 104. Client applications and other systems interact with and manage network elements 108a-108n in network element layer 104 through network manager 106.

Network manager 106 includes subsystem server 202. Subsystem server 202 is a software process or application that executes on network manager

106. Subsystem server 202 provides the interface for network manager 106 to all of the network entities, or subsystems, that are not part of the network manager 106. In operation, subsystem server 202 is an application running on a computer which acts as a clearinghouse for all of the messaging that goes on within the network management system 100. Subsystem server 202 provides an environment for the execution of managed objects 204. Managed objects 204 are a logical representation of a particular network element 108a-108n in the network management layer 102. Network manager 106 receives messages from client applications wishing to interact with or control the network. Network manager 106 passes these messages and to subsystem server 202. Subsystem server 202 examines a message received from network manager 106 and determines to which managed object 204 the message is to be routed.

Generally, managed objects 204 represent real-world managed network resources. Managed object class specifications define detailed characteristics which instances of that class may exhibit through behavior definitions, attributes, and actions. Behavior definitions are textual descriptions which describe how the managed object behaves and reacts and possibly how it is related to other managed objects. Attribute values represent data associated with the managed objects or relationships with other objects. Actions are operations that may be performed on managed object instances within the system. As with any object-oriented system, the concepts of encapsulation and data abstraction or inheritance are used. Inheritance, or the sharing of characteristics such as data and methods are based on a hierarchial relationship.

Managed object 204 comprises body 206 and characteristics 208. Characteristics 208 includes attributes 210, actions 212 or both. Body 206 controls the behavior of the managed object 204. Messages sent or received by managed object 204 with the network management system 100 are processed and generated by body 206. Body 206 receives, parses and distributes to characteristics 208 any messages received from network management system 100. Likewise, any message to be sent by managed object 204 are generated by body 206. Characteristics 208 represent the data and functionality that are available at managed object 204 or corresponding network element 108a-108n.

Assume, for the purposes of explanation, that managed object 204 represents a given network element 108a in the network element layer 104. The information stored by network element 108a would be represented in managed object 204 as characteristics 208. Characteristics 208 represent the

information (attributes 210) and functionality (actions 212) available at network element 108a. Characteristics 208 include all of the data members of the managed object class. In the example managed object of FIG. 2, characteristics 208 includes attribute characteristics 210 and action characteristics 212. For the purposes of explanation, attribute characteristics 210 and action characteristics 212 will be used interchangeably with attributes 210 and actions 212, respectively.

Attributes 210 are the data members in the managed object 204 or corresponding network elements 108a-108n available for retrieval and storage by network manager 106. Network manager 106 sets and retrieves attributes 210 by sending and receiving messages from network element 108a. Actions 212 represent functionality available to network manager 106 at managed object 204 or corresponding network elements 108a-108n. Network manager 106 sends command messages to network elements 108a-108n, causing the network elements 108a-108n to perform actions, and network elements 108a-108n respond with a response message confirming the execution of the action. Together, attributes 210 and actions 212 represent the characteristics 208 of managed object 204.

Managed objects 204 are not constantly running threads on the subsystem server. Managed objects 204 only execute, or exist, on subsystem server 202 when a message for a particular managed object 204 has been received at subsystem server 202, and while the managed object 204 is processing the message. When managed objects 204 are not active, they are stored on persistent store off the subsystem server 202. Examples of persistent store are magnetic or optical media, rom, or other permanent type storage. Storage of the managed objects 204 on persistent store during periods of non-use promotes efficient use of network manager resources. If the message received by subsystem server 202 is destined for a managed object 204 which is not in existence on the subsystem server 202, subsystem server 202 retrieves the desired managed object 204 from persistent store and passes the message to it. If the original message received from the external system by network manager 106 is directed to more than one managed object 204, subsystem server 202 parses the original message into pieces that are interpreted and acted upon by individual managed objects 204.

Subsystem server 202 includes at least one managed object 204. Each network element 108a-108n in network element layer 104 has an associated representative managed object 204 on subsystem server 202. The managed objects 204 on network manager 106, therefore, are a logical representation of all the network elements 108a-108n in network management layer 102.

Managed object 204 represents the attributes and actions available at each network element 108a-108n. For example, a managed object 204 will have data members corresponding to its exemplary network element 108a including the data transmission error rate, number of errored seconds and number of severely errored seconds of the network element 108a.

In the example of FIG. 2, only one managed object 204 is shown in subsystem server 202. In reality, there are at least as many managed objects 204 on subsystem server 202 as there are network elements 108a-108n. Managed objects 204 provide a logical interface at network manager 106 to the network element layer 104. The interface provides a means for the network manager 106 to communicate with, retrieve data from, and cause actions to be performed in the network element layer 104. In addition to managed objects 204 representing network elements 108a-108n, additional managed objects 204 may reside on subsystem server 202 to represent management functions available to users, applications and other systems interacting with network manager 106.

The preferred embodiment of the relationship between managed object 204 and network elements 108a-108n is a managed object tree. Managed object trees are tree structures built from individual managed objects arranged in a hierarchical structure. Features and functionality of network element 108a-108n is built up from multiple managed objects 204 resulting in the total set of features and functionality encompassed by a network element 108a-108n. Details and examples of managed object trees are described in more detail in conjunction with FIG. 6, below.

FIG. 3 illustrates the format of an exemplary message 300. Message 300 is an example of a message to a managed object 204, transmitted between network manager 106 and another network entity, including a user, application or system external to network management system 100. The format of message 300 is illustrated for the purposes of example. Message formats different from those described herein would be contemplated by a skilled artisan in the field of network management.

Exemplary message 300 includes header 316 and payload 318. Header 316 uniquely identifies the particular message and message type. Header 316 includes object ID (OID) 302, transaction ID (XID) 304, and message type 306. OID 302 identifies the particular managed object 204 for which message 300 is intended. For example, a message 300 intended for, or related to, network element 108a would have an OID 302 identifying the managed object 204 corresponding to a given network element 108a. Each managed object has a unique OID 302. In the preferred embodiment, OID 302 is an 80-bit pure

number. If managed object 204 were to generate a message for transmission to network element 108a, the one-to-one correspondence between managed objects 204 and the network elements 108a-108n would allow the address of network element 108a to be algorithmically computed directly from OID 302.

XID 304 identifies the particular transaction with which message 300 is associated. For example, a message from a user external to network management system 100 to network manager 106 would have a unique XID 304. The response message from network manager 106 to the user would use the same unique XID 304, or an XID 304 which was algorithmically determinable from the original XID 304. The XID 304 provides for the identification and management of transaction and transaction response messages among the entities in network management system 100. Unique XIDs 304 guarantee that concurrent messages do not result in corruption of the information passed between the network entities during network management.

Message type 306 indicates the nature of the operation requested or specified by message 300. For example, message 300 may be of message type 306 indicating a get attribute request. A get attribute request is a request to retrieve data members stored by attributes 210. A get attribute message requests that managed object 204 respond with the value of attributes 210. Conversely, message type 306 may specify a set attribute request. A set attribute message is a request to set (or reset) the data values of attribute 210. Other examples of message types 306 of the preferred embodiment are a response to a get message type, a response to a set message type, an action request message type, a response to an action request message type, create and destroy managed object message types, etc.

An action message type requests that an action be performed at managed object 204. Since managed object 204 is a logical representation of network element 108a-108n, an action request message usually specifies actions to be performed at network element 108a-108n. In such cases, managed object 204 will send an additional action request message to its corresponding network element 108a-108n. For example, a lock action message type is provided to network management system 100 to bar a particular resource from providing a service. The network element specified in the lock action message is essentially "locked out" in response to a lock action message. An unlock action message type results in a locked network element or resource becoming "unlocked." Responses to action requests are usually action status messages. An action status message is a response to an action request

message and indicates that the action request was completed or was not completed.

Create and destroy message types create and destroy managed objects 204 on network manager 106. Managed objects 204 are created and destroyed on network manager 106 in order perform the process of software version management. New managed objects are created to implement new software functionality, and old managed objects are destroyed to remove redundant or outdated software. For example, if network element 108a is added to network element layer 104, a corresponding managed object 204 must be created with a managed object create message on network manager 106. Managed objects 204 on network manger 106 are also created and destroyed to implement and remove additional functionality at the network management layer 102. The create managed object message is sent from the client external to network manager 106 to create managed object 204. Alternatively, part of the initialization process of the network element 108a may send a create managed object message to network manager 106. If network element 108a is removed from network element layer 104, the corresponding managed object in network management 106 is removed with a destroy managed object message type.

Message 300 further includes payload 318 comprising identifier (ID) and data 308a-308n. Payload 318 identifies the particular characteristics 208 by which message 300 is to be implemented. ID and data 308a-308n are used by body 206 to generate message element 400 as described below. For example, when managed object 204 receives a set attribute message, message 300 will change data value(s) in attributes 210.

ID information in ID and data 308a-308n identifies the particular characteristic 208 to which the message 300 is directed. For example, suppose message 300 is received by network manager 106. OID 302 identifies the particular managed object 204 to which message 300 is to be routed. Message 300 is routed by subsystem server 202 to managed object 204. Managed object 204 receives message 300. Body 206 of managed object 204 examines ID and data information 308a-308n of payload 318. Body 206 parses message 300 into smaller "message elements" to be passed to characteristics 208. Message type 306 of header 316 and ID and data 308a-308n of payload 318 are passed to characteristic 208 corresponding to ID in ID and data 308a-308n of payload 318.

FIG. 4 illustrates the format of an exemplary message element 400. Message element 400 represents the information transferred between body 206 of managed object 204 and characteristics 208. Body 206 receives message

300 and parses it into at least one message element 400. Message element 400 includes an internal ID 402, message type 306 and data 404. Internal ID 402 is determined from ID information in ID and data fields 308a-308n of payload 318. Internal ID 402 identifies the particular data member in characteristics 208. For example, internal ID 402 may identify attribute characteristic 210, and an associated data value within attribute characteristic 210. The data value within attribute 210 may represent a feature, or a function in the network element 108a-108n associated with managed object 204.

Internal ID 402 targets message element 400 for a particular data member within a particular attribute 210 or action 212 in characteristics 208. Message type 306 within message element 400 is copied from message type 306 within message 300. Message type 306 within message element 400 identifies the type of request or command associated with message element 400. For example, message type 306 may identify message element 400 as a get attribute message. In such an example, body 206 would route message element 400 to attribute characteristic 210. In the case of a get message element, the data field of the message element would contain no data (the data is to be retrieved). Internal ID 402 identifies the particular data member within attribute 210 that the get message element is requesting.

Attribute characteristic 210 responds to the get message element by generating a response message element including the values of the data members identified by internal ID 402 in attribute characteristic 210. The response message element is sent to body 206 of managed object 204. The response message element includes a message type 306 of get response, indicating that the response message element is a response to a get type message element. Internal ID 402 of the response message element 400 is unchanged, thereby identifying the response message element 400 to body 206 as a response to the original get type message element 400. In an alternative embodiment, internal ID 402 of the response message element may be algorithmically determined from the get attribute message element Internal ID 402.

FIGS 3. and 4 illustrate exemplary embodiments of message 300 and message element 400. It should be noted, however, that one of the features of the present invention is the flexibility with which messaging systems, or protocols, can be accommodated. If, for example, the network management system 100 includes a plurality of hierarchical levels, the protocol of the messages and message elements can be changed to accommodate the nature of the network entities that will be receiving the messages.

Characteristics 208 on managed object 204 determine the nature of the messages to be sent from managed object 204 to network elements 108a-108n. For example, if managed object 204 receives a get attribute request message, the message is parsed into message elements 400 and passed to the appropriate attribute 210 in managed object 204. If attribute 210 is associated with particular data values stored by network element 108a, managed object 204 must send a get attribute request message to network element 108a in order to retrieve the data associated with the get attribute request message received by managed object 204.

Network element 108a receives the get attribute request message from network manager 106 and responds by sending a get attribute response message containing the data requested to network manager 106. The get attribute response message is passed to managed object 204, where the get response attribute response message is parsed into message elements 400 as described above. Attribute characteristic 210 is updated with the data 404 in message element 400 from network element 108a. After attribute characteristic 210 is updated with data 404 from message element 400, attribute characteristic 210 sends a response message element to body 206. Body 206 translates the response message element into a message containing the data to be sent to the requesting client application.

FIG. 5 further illustrates an exemplary network element 108a. Network element 108a comprises managed object 502 which in turn comprises body 504 and characteristics 506. Characteristics 506 includes attributes 508, actions 510 or both. Body 504 controls the behavior of the managed object 502. Messages sent or received by managed object 502 to network management system 100 are processed and generated by body 206. Body 206 receives, parses and distributes to characteristics 208 any messages received from network management system 100. Likewise, any message sent by managed object 502 is generated by body 504. Characteristics 506 represent the data and functionality that is available at network element 108a-108n.

Managed object 502 interacts with the hardware of network element 108a. For example, message 300 is received from network manager 106 at network element 108a. OID 302 identifies the particular network element 108a and managed object 502 to which message 300 is directed. Managed object 502 parses received message 300 into message elements 400, at body 504. Characteristics 506 respond to received message element 400 passed to them by body 504 by executing an action at the network element (e.g., switching a digital cross-connect, changing the configuration of the network element, etc.) or setting data attributes 508 with the data 404 in message 400.

Alternatively, message 300 received at network element 108a may request information owned by network element 108a. In such instances, managed object 502 will formulate a response message 300 and transmit it over the network to network manager 106.

Software Versioning

Upgrading is the process of changing from an older first software version to a newer second software version at network manager 106. Usually, the need for an upgrade to the software on network manager 106 is driven by changes in the hardware or software configuration of network elements 108a-108n themselves. In such a case, the interface that network elements 108a-108n present to network manager 106 changes. This change in network element interface necessitates a version change in software at network manager 106, as well as a change in software at network elements 108a-108n. In the present invention, the procedure for upgrading software on network manager 106 is to first upgrade the network manager 106 software, and then upgrade the network elements 108a-108n. This method of upgrading the software on network management system 100 is referred to as "top-down."

The software version management process of the present invention is top-down. The top-down software version management process upgrades the network manager first, and needs to support two versions of software, while the network elements are being upgraded. At any point in time during an upgrade, some network elements will be running the old software version and some network elements will be running new software version. After the upgrade, all of the subsystems running the old software version have been upgraded to the new software version. After the network manager has been upgraded, the network manager migrates the network elements from the old software version to the new software version one by one. After the network manager has upgraded all of the network elements, the upgrade is complete.

Network element upgrade is the process of moving a particular network element from an older first software version to a newer second software version. In the preferred embodiment, network element upgrade is only completed after a successful upgrade of network manager 106. Alternative embodiments, however, contemplate concurrent upgrade of the software of network manager 106 and network elements 108a-108n. Alternatively, the network element software may be upgraded prior to the completion of the upgrade of the network manager software.

After each of the network elements 108a-108n has been upgraded to the new software version, the upgrade is complete. At any point in time during an upgrade, some network elements 108a-108n will be running the old software version and some network elements 108a-108n will be running the new software version. After the upgrade, all of the network elements which required an upgrade will be running the new software version.

FIG. 6 illustrates the preferred embodiment of a managed object tree 600. Managed object tree 600 is a hierarchial structure of managed objects 602a-602n. In the preferred embodiment, each network element 108a-108n has an associated managed object tree 600. Each managed object tree 600 on a network element 108a-108n has a corresponding managed object tree 600 on network manager 106. Network element 108a-108n functionality is built into the hierarchial structure of managed object tree 600. In the case of a managed object tree 600 associated with network element 108a-108n, managed objects 602a-602n represent network element functionality and features of each network element 108a-108n with which they are associated. Managed objects 602a-602n are arranged in a parent-child relationship. Parent-child relationships define the hierarchical structure of managed object tree 600. For example, managed objects 602d and 602e are children of managed object 602b. Each of child managed objects 602d, 602e of managed object 602b has associated functionality and features subordinate to functionality and features of parent managed object 602b.

For example, network element 108a may contain a shelf of telecommunications circuit packs. Managed object tree 600 associated with network element 108a is arranged in a hierarchial structure of managed objects 602a-602n representing the functionality, component parts and features of network element 108a. Managed object 602a, or the "root" managed object, identifies managed object tree 600 as being associated with network element 108a.

In such an example, child managed object 602b may represent a particular shelf of network element 108a. Managed object 602b defines the parameters and functionality of the particular shelf with which it is associated. Managed objects 602d and 602e are children of shelf managed object 602b. Managed objects 602d and 602e may represent circuit packs on the shelf represented by managed object 602b. Managed object 602d may represent single or multiple circuit packs of a particular kind. Managed object 602e may represent single or multiple circuit packs, of a type the same or different from managed object 602d. Each telecommunications circuit pack on the shelf would be associated with a child managed object of managed object

602b. In the instant example, root managed object 602a identifies managed object tree 600 and provides the base functionality of the managed object tree. Root managed object 602a identifies the particular network element 108a-108n with which managed object tree 600 is associated. Root managed object 602a is the "parent" of managed objects 602b-602n in managed object tree 600.

The preferred embodiment of the present invention comprises a naming convention to identify and position managed objects 602a-602n in managed object tree 600. The naming convention is a scheme to uniquely identify the managed objects of network management system 100. The preferred embodiment of the naming convention is the Object ID (OID) 302. Each managed object has an associated OID 302. In the preferred embodiment, the upper 8-bits of OID 302 identify the particular network element with which a managed object is associated. All of managed objects 602a-602n necessarily have the same upper 8-bits of OID 302 which identify the particular network element 108a-108n associated with managed object tree 600. The upper 8-bits of the OID 302 is referred to as the agent ID. The agent IDs of managed objects 602a-602n are the same.

OID 302 also comprises the fixed domain name (FDN). The FDN has a class identifier, version identifier, and name. The class identifier identifies the class to which a particular managed object belongs. As is known in the art, an object class is a template for defining the methods and variables for a particular type of managed object. All managed objects of a given class are identical, or extremely similar in form and behavior but contain different data in their variables. The version identifier identifies the software version of the managed object. The name is an instance variable that stores a unique value to identify the particular instance of the class represented by the managed object. When a managed object is upgraded from one version to another, the class identifier will not change if the managed object has not changed in functionality (i.e., the managed object is an instantiation of the same class). If the managed object has changed in functionality, the class will change to denote a change in the managed object itself. From one version to another, the version identifier will change to denote that the managed object is at a different software version. The new version of the managed object tree 600 also has a new, unique agent ID. The new agent ID identifies the upgraded managed object tree 600 as the new managed object tree. The new and old agent IDs make switching over from the old managed object tree to the new managed object tree possible. From one version to another, each of the managed objects 602a-602n in a managed object tree 600

will acquire new names which uniquely identify those particular managed objects.

The FDN of the preferred embodiment also defines a complete path to managed objects 602a-602n in managed object tree 600 from the top of managed object (or root) tree 600 to the managed object's position. For example, managed object tree 600 of FIG. 6, comprises managed objects 602a-602n, each of which has a fully distinguished name. For the purposes of explanation, each of the managed objects of FIGs. 6-8 is labeled with its class (upper case letters of the alphabet), and its name (lower case version of its class).

An exemplary fully distinguished name for root managed object 602a is "Root-1-root." "Root" in "Root-1-root" identifies managed object 602a as belonging to class Root. The "1" in "Root-1-root" identifies managed object 602a as software version 1. The "root" in "Root-1-root" identifies managed object 602a as having the name "root." Since managed object 602a, "Root-1-root" is not a child of any other managed object, "Root-1-root" is its complete FDN.

In the example of FIG. 6, managed object 602b has an FDN expressed as "Root-1-root:A-1-a." "A" is the class, "1" is the version and "a" is the name of managed object 602b. The class, version and name of managed object 602b is prefaced by the "path" of managed objects traversed to the position of managed object 602b. Since managed object 602b is a child only of managed object 602a, only managed object 602a is included in the path. Similarly, the FDN path expressions for managed objects 602c-602f are as follows: "Root-1-root:B-1-b;" "Root-1-root:A-1-a:C-1-c;" "Root-1-root:A-1-a:D-1-d;" "Root-1-root:A-1-a:C-1-c:E-1-e;" and "Root-1-root:A-1-a:C-1-c:F-1-f," respectively.

FDN regular expressions (FDNREs) are expressions formed from FDNs. FDNREs are evaluated to select managed objects that satisfy the FDNRE. FDNREs are evaluated by network manager 106 to determine the particular managed objects 602a-602n identified by the FDNRE. For example, "Root-1-root:A-1-a:*" selects all of the child managed objects of 602b, namely 602d, 602f, and 602n. FDNREs are useful in the creation of expressions which evaluate out to entire managed object trees, or children of managed objects.

FIG. 7 illustrates an example of managed object tree topology transformation. Topology is the arrangement of managed objects in a managed object tree. Topology transformation is the process of transforming one managed object tree topology to another. In software version management, old software version managed object tree 702 is transformed into new software version managed object tree 704. After

migration, two copies of managed object tree 702 exist on network manager 106. Topology transformation transforms the copy of managed object tree 702 into managed object tree 704. The exemplary managed object trees 702, 704 of FIG. 7 represent different software versions of managed object trees 702, 704 associated with the same network element.

Network manager 106 contains two domains, the containment domain and operational domain. The containment domain stores active data defining rules by which managed objects in network manager 106 can form parent-child relationships. The containment domain and operational domain are logical divisions of subsystem server 202. The preferred embodiment of the active data in the containment domain is managed objects. The operational domain is a domain in which the managed objects are system software on network manager 106 associated with network elements 108a-108n. The containment domain, therefore, establishes the rules by which operational domain managed objects are organized.

Before the upgrade procedure can be performed, the network manager 106 must be primed with new and old containment hierarchies. The active data is stored in the containment domain as managed objects. The containment domain, therefore, establishes the rules by which managed objects on network manager 106 are organized. The containment domain rules are established specifications for particular software versions. For example, a containment domain managed object may specify that circuit pack managed objects may not have shelf managed object children.

FIG. 8 will be used in conjunction with FIGs. 9-11 to explain the software versioning process. FIG. 8 illustrates containment domain 802 and operational domain 804. Both domains include managed object trees 806, 810, 814 and 818. Containment domain 802 comprises managed object trees 806 and 808 and operational domain 804 comprises managed object trees 814 and 818. Managed object trees 806 and 810 are referred to as containment domain trees and managed object trees 814 and 818 are referred to as operational domain trees.

Containment domain 802 manages operational domain 804. Containment domain trees 806 and 810 comprise managed objects 808a-808n and 812a-812n, respectively, which stipulate the topology of operational domain trees 814, 818 in operational domain 804. This relationship is illustrated by the arrows drawn between managed object tree 806 and 814, and the arrow between managed object tree 810 and managed object tree 818. Containment domain managed object trees 806, 810 are specified and defined for the network management system 100 for each release of software.

Containment domain managed object trees 806, 810 provide the specification for "legal" combinations of managed objects in managed object trees in the system. Each new software version includes new containment domain managed object trees defining new combinations of managed objects for new configurations of operational domain managed object trees. A containment domain managed object tree may specify the topology for a plurality of operational domain managed object trees.

Furthermore, each managed object in containment domain 802 may be associated with multiple managed objects in operational domain 804 because a managed object in containment domain 802 specifies the rules for instantiation of managed objects in operational domain 804. FIG. 8 illustrates one such relationship. Containment domain managed object 808n specifies the rules for instantiation of managed objects of class "F." Managed object tree 814 in operational domain 804 has two managed objects 816g and 816n of class "F." One containment domain managed object 808n, therefore, is associated with two operational domain managed objects 816g, 816n.

Managed object tree 806 includes the rules of construction for operational domain tree 814. The rules of containment domain managed object tree 806 specify "legal" parent-child relationships for the managed object trees in the operational domain. The managed objects 808a-808n of managed object tree 806 specify the number and type of legal combinations of managed objects in operational domain managed object tree 814. These containment domain managed object rules specify, for example, cardinality, class, type, and parent-child relationships of operational domain managed objects 816a-816n. Two versions of software are represented by managed object trees 806, 810, 814 and 818. Managed object trees 806 and 814 represent an older, first software version, and managed object trees 810 and 818 represent a newer, second software version. In the example of FIG. 8, each of operational domain 804 managed object trees 814, 818 represents the same network element in network element layer 104. Since the managed object tree pairs of FIG. 8 represent the same network element, both managed object trees 806, 810 in containment domain 802 have the same agent ID.

There are several major steps to software versioning. The first step is loading new containment domain managed object tree 810 into containment domain 802. Containment domain managed object tree 810 is provided by a user or software application external to network manager 106. Containment domain managed object tree 810 specifies the valid hierarchical structure managed object tree 818. Next, the process of topology

transformation creates new operational domain managed object tree 818. To create the new managed object tree 818, operational domain managed object tree 814 is processed by a set of topology transformation rules, resulting in new software version managed object tree 818. After new software version managed object tree 818 has been created, the data contained in each of managed objects 820a-820n is evolved. After the data is evolved, the new operational domain managed object tree 818 is complete. Software is then migrated to the associated network elements 108a-108n.

After the data has been evolved from old managed object tree 814 to new managed object tree 818, network manager 106 downloads the new software to the associated network elements. Network manager 106 then uncouples old managed object tree 814 from network element 108a and recouples new managed object tree 818 to network element 108a. After network element 108a has been recoupled, the software versioning process is complete.

The process of software versioning requires a number of inputs. The first input is the new software version containment domain managed object tree 810. Another input is the topology transformation rule base. The topology transformation rule base is a file, preferably a database, containing a set of conditional operators which govern the creation and destruction of managed objects during topology transformation. The topology transformation rule base is a set of operators, each associated with the creation or deletion of managed objects in managed object trees.

The last input is the data evolution rule base. The data evolution rule base is a file, preferably a database, of functions for each managed object 816a-816n that requires data transformation. Topology transformation results in a new managed object tree. The managed objects in the new managed object tree, however, may require upgrading individually if their data structure has changed from the old software version to the new software version. A data evolution function is essentially a script that performs data transformation on a managed object. Each managed object 816a-816n that requires data evolution from the old software to the new will have an associated data evolution function. An additional input into the process of software versioning is the operational domain managed object tree 814 corresponding to the old software version.

FIG. 8 illustrates old operational domain managed object tree 814 and new operational domain managed object tree 818. The following seven FDN statements describe the topology transformation of managed object tree 814 into managed object tree 818:

Root-1-root _ Root-2-root; (1)
Root-1-root:A-1-a _ Root-2-root:A-2-a; (2)
Root-1-root:B-1-b _ Root-2-root:B-2-b; (3)
Root-1-root:A-1-a:C-1-c _ Root-2-root:B-2-b:C-2-c; (4)
Root-1-root:A-1-a:C-1-c:E-1-e _ Root-2-root:B-2-b:C-2-c:E-2-e; (5)
Root-1-root:A-1-a:C-1-c:F-1-f _ Root-2-root:B-2-b:C-2-c:F-2-f; (6)
Null _ Root-2-root:B-2-b:C-2-c:G-2-g. (7)

The portion of the FDN statements to the left of the "_" is the FDN for managed objects 816a-816n in the old software version managed object tree 814. The "_" represents a topology transformation rule, and the portion of the statement to the right of the "_" is a new FDN for the resultant managed object(s) in the new software version managed object tree. FDN statements 1-7 are examples of the types of statement adapted for topology transformation from one software version to another.

In the example FDN statements 1-7 above, the FDN for managed objects 816a-816n identifies a particular managed object 816a-816n and its location in managed object tree 814.

The topology transformation rule transforms the particular managed object 816a-818n into managed object 820a-820n specified by the FDN for managed objects 820a-820n. The FDNs for new managed objects 820a-820n are specified by the topology transformation rule associated with the FDN for each old managed object 816a-816n. Essentially, each of managed objects 816a-816n is selected, looked up in the topology transformation rule base and transformed per the topology transformation rule and FDN(s) specified.

The topology transformation rule base comprises four main types of topology transformation rules. A type one topology transformation rule handles the transformation of a managed object that is deleted from one version to another. For example, during topology transformation a managed object may be deleted if it is no longer supported in the new software version. Such a managed object would be deleted during topology transformation.

A type two topology transformation rule transforms a single managed object into more than one managed object in the new managed object tree. In such a case, a single managed object in an operational domain managed object tree becomes a plurality of managed objects during topology transformation. A single managed object representing a plurality of circuit packs in the old software version, and a managed object for each circuit pack in the new software version is an example of such a transformation. If a

managed object tree in the old version has a single managed object representing a plurality of circuit packs, the new managed object tree will have a plurality of managed objects representing the circuit packs in the managed object tree.

A type three topology transformation rule creates a managed object which does not exist in the old software version and has to be created in the new managed object tree. New managed object types, network element functionality or configuration may require the instantiation of new managed objects.

A type four topology transformation rule transforms a plurality of managed objects into one. An example of such a case is the reverse of the circuit pack example above. If a managed object tree in the old version has a managed object representing individual circuit packs, the new managed object tree will have a single managed object representing a plurality of circuit packs. The plurality of managed objects representing individual circuit packs will be coalesced into a single managed object representing a plurality of circuit packs.

The FDN statements establish the topological transformation from old managed object tree 814 to new managed object tree 818. The default FDN statement for each of the managed objects is that they occupy their same topological position in the new managed object tree 1104. If the topology transformation rule base does not specify particular positions in the new managed object tree 818 for the old managed objects 816a-816n, the new managed objects 820a - 820n are placed in the same position as they occupy in old managed object tree 814. Only those managed object topologies that have changed need be specified in the topology transformation rule base. This system of default rules for topological transformation of managed object trees expedites the topological transformation of managed object trees from one version to another.

FIG. 9 illustrates an overview of the software versioning process. FIG. 9 will be explained in terms of the managed objects and managed object trees 806, 810, 814 and 818 of FIG. 8. Step 902 determines the FDN of the managed object 816a-816n to be upgraded from a first software version to a second software version. The FDN may be specified by the user, a control program or other initialization means. The FDN may be specified directly, or calculated from the agent ID of a network element to be upgraded.

Although the process of the present invention is described in terms of upgrading the managed object software for single managed object, or managed object tree, the present invention may be extended to upgrading

managed object software for multiple network elements concurrently. Similarly, the present invention contemplates upgrading individual managed objects and managed object trees, multiple managed objects and managed object trees, and supporting multiple software versions at network elements 108a-108n and network manager 106. Step 904 identifies the managed object tree 814 associated with the FDN identified in step 902. For example, if the FDN determined in step 902 is "Root-1-root:A-1-a:C-1-c:F-1-f" which is associated with managed object 816g, step 904 identifies managed object tree 814.

Step 906 performs the topology transformation of managed object tree 814. Step 906 traverses managed object tree 814, and builds a list of FDNs associated with managed objects 816a-816n. In a preferred embodiment, an FDNRE associated with managed object tree 814 is evaluated to yield the list of managed objects 816a-816n. Each of the FDNs are used to search the topology transformation rule base for topology transformation rules associated with managed objects 816a-816n.

Step 908 validates new managed object tree 818 topology. Containment domain managed object trees 806, 810 provide the specification for "legal" combinations of managed objects in managed object trees in the system. Step 908 checks the topology of managed object tree 818 against containment domain managed object tree 810 to verify that the new topology is valid. If step 908 determines that managed object tree 818 has an invalid topology, an error is raised, and the software version management process is stopped.

Step 910 performs data transformation for each of managed objects 820a-820n of managed object tree 818. Data transformation is the process of transforming the data contained in managed objects 820a-820n from the old software version to the new software version. Topology transformation builds new managed object tree 818, but managed objects 820a-820n have not changed. Data transformation provides for changes to the data contained in managed object 820a-820n.

Data transformation is performed by data transformation rules stored in a data transformation file, preferable a database. Data transformation rules are functions defined by a user or other outside source that transform the data members of managed objects. The data transformation rule base is searched for each of the managed objects 820a-820n in managed object tree 818. If a data transformation rule is found, the function is performed, thereby transforming the data members of the particular managed object.

After step 910, managed object tree 818 is ready for implementation at network element layer 104. Step 912 uncouples network element 108a-108n

from its associated managed object tree 814 and couples it to managed object tree 818. After network element 108a-108n has been recoupled to new managed object tree 818, the upgrade process for the instant network element is completed.

FIG. 10 further illustrates step 906, topology transformation of managed object tree 814. The process of step 906 begins at step 1002. Step 1002 retrieves or creates a new FDNRE for new software version managed object tree 818. The new FDNRE may be specified by an outside user or a software upgrade procedure. In an alternative embodiment, the new FDNRE is specified by the topology transformation rule associated with root managed object 816a. The new FDNRE identifies new managed object tree 818.

Step 1004 traverses old software version managed object tree 814 and builds a list of managed objects 816a-816n. Step 1004 evaluates an FDNRE identifying managed object tree 814 and its associated managed objects 816a-816n. The FDNRE evaluated in step 1004 identifies the managed object tree 814 for upgrading. Step 1004 builds a list of all managed objects 816a-816n and their associated FDNs. Step 1006 sorts the managed object list built in step 1004 to ensure parent/child relationships. The managed object list is sorted because the order of managed objects 816a-816n in the list determines the order in which new software version managed object tree 818 is built.

Step 1008 searches the topology transformation rule base to find a topology transformation rule associated with each FDN in the object list from step 1006. The topology transformation rule specifies the particular process by which the managed object from the old software version managed object tree will be transformed. The four types of topology transformation rules are described above.

Step 1010 performs the process of topology transformation by processing the list of managed object FDNs and their associated topology transformation rules. Step 1010 is further explained in conjunction with FIG. 11 below. Step 1010 results in new software version managed object tree 818.

Step 1012 determines if all managed objects in new software version managed object tree 818 were created successfully. This check can be performed in a number of ways, examples of which are log files built during the topology transformation process or error files built during the evaluation of topology transformation rules. If step 1012 determines that not all managed objects were created successfully, step 1014 deletes the new managed object tree and returns an error to the user. If step 1012 determines

that all managed objects were created successfully, the process of step 906 continues at step 910.

FIG. 11 further illustrates step 1010, the process of topology transformation by processing the list of managed object FDNs and their associated topology transformation rules. Although FIG. 11 illustrates topology transformation for a single managed object and its associated FDN, the process of FIG. 11 is repeated for each of the managed objects 816a-816n in managed object tree 814. The preferred embodiment of the present invention is to process all of the managed objects 816a-816n in the list generated in step 1006.

The process of topology transformation begins at step 1102. Step 1102 receives the rule associated with the FDN for the current managed object. Each of the FDNs for the managed objects determined in step 1006 are processed in the topology transformation steps 1102-1126 of FIG. 11. Each of the managed objects and associated FDNs determined in step 1006 are processed by their associated rule from the topology transformation rule base in the process of FIG. 11. Step 1104 determines if the rule associated with the instant managed object is a type one rule. A type one rule indicates that a managed object that exists in managed object tree 814 does not exist in managed object tree 818. The managed object, therefore, is not created in the new managed object tree. If step 1104 determines that the topology transformation rule is a type one rule, the managed object is not created in the new managed object tree at step 1106. After step 1106, the process of step 1010 terminates at step 1128. If step 1104 determines that the rule is not a type one rule, the process of step 1010 continues at step 1108.

Step 1108 determines if the topology transformation rule is a type two topology transformation rule. A type two topology transformation rule transforms a single managed object into more than one managed object in the new managed object tree. If step 1108 determines that the rule is a type two rule, the process of step 1010 continues at step 1110. Step 1110 creates the new managed objects in the new managed object tree. The number of new managed objects created in the new managed object tree corresponds to the list of new FDNs specified by the type two rule in the topology transformation rule base. After step 1110 creates the new managed object in the new managed object tree, the process of step 1010 terminates at step 1128. If step 1108 determines that the rule is not a type two rule, the process of step 1010 continues at step 1112.

Step 1112 determines if the topology transformation rule is a type four topology transformation rule. A type four topology transformation rule

transforms a plurality of managed objects into one. If the topology transformation rule is a type four rule, step 1114 determines whether a new managed object in managed object tree 818 already exists that is associated with the FDN of the old software version managed object. If step 1114 determines that the managed object exists in the new software version managed object tree 818, the process continues to step 1120 and a new managed object is not created in managed object tree 818. If step 1114 determines that the managed object does not yet exist in managed object tree 818, step 1116 creates a new managed object in managed object tree 818. After step 1116 the process terminates at step 1128.

Step 1118 searches the topology transformation rule base for type three rules associated with new software version managed object tree 818. A type three topology transformation rule creates a managed object which does not exist in the old software version and has to be created in the new managed object tree. Since the managed objects do not exist in managed object tree 814, there are no old managed object FDNs associated with type three rules. Step 1122 determines whether step 1118 found a type three rule associated with the current managed object tree. If step 1122 determines that a type three rule for the current managed object tree has been found, step 1124 creates a new managed object in managed object tree 818. Step 1124 creates the new managed object with an FDN specified by the type three rule in the topology transformation rule base. If step 1122 does not find a type three rule, the process of step 1010 continues at step 1126.

Step 1126 indicates that no topology transformation rules associated with the old software version managed object FDN were found in the topology transformation rule base. When no topology transformation rules are associated with a managed object in managed object tree 814, software version management uses the default rule to create a managed object of the same type in the new managed object tree. The default rule of step 1126 creates a new managed object of the same type and child-parent relationship as the managed object in the old software version managed object tree.

The default rule of step 1126 allows a minimum number of topology transformation rules to be stored in the topology transformation rule base. Since rules are only added to the topology transformation rule base when the topology of a managed object tree has changed, software versions without extensive changes to previous software versions are stored in a minimal amount of space in the topology transformation rule base.

The present invention may be implemented using hardware, software or a combination thereof and may be implemented in a computer system or

other processing system. In fact, in one embodiment, the invention is directed toward a computer system capable of carrying out the functionality described herein. An example computer system 1201 is shown in FIG. 12. The computer system 1201 includes one or more processors, such as processor 1204. The processor 1204 is connected to a communication bus 1202. Various software embodiments are described in terms of this example computer system. After reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

Computer system 1202 also includes a main memory 1206, preferably random access memory (RAM), and can also include a secondary memory 1208. The secondary memory 1208 can include, for example, a hard disk drive 1210 and/or a removable storage drive 1212, representing a floppy disk drive, a magnetic tape drive, an optical disk drive, etc. The removable storage drive 1212 reads from and/or writes to a removable storage unit 1214 in a well known manner. Removable storage unit 1214, represents a floppy disk, magnetic tape, optical disk, etc. which is read by and written to by removable storage drive 1212. As will be appreciated, the removable storage unit 1214 includes a computer usable storage medium having stored therein computer software and/or data.

In alternative embodiments, secondary memory 1208 may include other similar means for allowing computer programs or other instructions to be loaded into computer system 1201. Such means can include, for example, a removable storage unit 1222 and an interface 1220. Examples of such can include a program cartridge and cartridge interface (such as that found in video game devices), a removable memory chip (such as an EPROM, or PROM) and associated socket, and other removable storage units 1222 and interfaces 1220 which allow software and data to be transferred from the removable storage unit 1222 to computer system 1201.

Computer system 1201 can also include a communications interface 1224. Communications interface 1224 allows software and data to be transferred between computer system 1201 and external devices. Examples of communications interface 1224 can include a modem, a network interface (such as an Ethernet card), a communications port, a PCMCIA slot and card, etc. Software and data transferred via communications interface 1224 are in the form of signals which can be electronic, electromagnetic, optical or other signals capable of being received by communications interface 1224. These signals 1226 are provided to communications interface via a channel 1228. This channel 1228 carries signals 1226 and can be implemented using wire or

cable, fiber optics, a phone line, a cellular phone link, an RF link and other communications channels.

In this document, the terms "computer program medium" and "computer usable medium" are used to generally refer to media such as removable storage device 1212, a hard disk installed in hard disk drive 1210, and signals 1226. These computer program products are means for providing software to computer system 1201.

Computer programs (also called computer control logic) are stored in main memory 1206 and/or secondary memory 1208. Computer programs can also be received via communications interface 1224. Such computer programs, when executed, enable the computer system 1201 to perform the features of the present invention as discussed herein. In particular, the computer programs, when executed, enable the processor 1204 to perform the features of the present invention. Accordingly, such computer programs represent controllers of the computer system 1201.

In an embodiment where the invention is implemented using software, the software may be stored in a computer program product and loaded into computer system 1201 using removable storage drive 1212, hard drive 1210 or communications interface 1224. The control logic (software), when executed by the processor 1204, causes the processor 1204 to perform the functions of the invention as described herein.

In another embodiment, the invention is implemented primarily in hardware using, for example, hardware components such as application specific integrated circuits (ASICs). Implementation of the hardware state machine so as to perform the functions described herein will be apparent to persons skilled in the relevant art(s).

In yet another embodiment, the invention is implemented using a combination of both hardware and software.

While the invention has been particularly shown and described with reference to preferred embodiments thereof, it will be understood by those skilled in the relevant art that various changes in form and details may be made therein without departing from the spirit and scope of the invention.

WHAT IS CLAIMED IS:

CLAIMS

1. In a network management system having a network manager and at least one network element, wherein the network manager has a plurality of managed object trees, a method for software versioning, comprising the steps of:

identifying a first managed object tree having at least one managed object to be upgraded;

creating a second managed object tree having at least one managed object by transforming the topology of said first managed object tree; and

performing data transformation of said at least one managed object of said second managed object tree.

2. The method of claim 1, wherein said step of creating further comprises traversing said first managed object tree and building a list of said at least one managed object of said first managed object tree.

3. The method of claim 1, wherein said step of creating further comprises identifying a topology transformation rule associated with said at least one managed object of said first managed object tree.

4. The method of claim 3, wherein said step of creating further comprises generating at least one managed object of said second managed object tree in response to said step of identifying.

5. The method of claim 3, wherein said step of creating further comprises determining the type of said topology transformation rule.

6. A computer program product for enabling a processor in a computer system to perform software versioning in a network management system having a network manager and at least one network element, wherein the network manager has a central processing unit (CPU), an operating system and a plurality of managed object trees, said computer program product comprising:

a computer usable medium having computer readable program code means embodied in said medium for causing an application program to

execute on the computer system, said computer readably program code means comprising:

a computer readable first program code means for causing the processor to identify a first managed object tree having at least one managed object to be upgraded;

a computer readable second program code means for causing the processor to create a second managed object tree having at least one managed object by transforming the topology of said first managed object tree; and

a computer readable third program code means for causing the processor to perform data transformation of said at least one managed object of said second managed object tree.

7. The computer program product of claim 6, wherein said second program code means further comprises program code means for causing the processor to traverse said first managed object tree and building a list of said at least one managed object of said first managed object tree.

8. The computer program product of claim 6, wherein said second program code means further comprises program code means for causing the processor to identify a topology transformation rule associated with said at least one managed object of said first managed object tree.

9. The computer program product of claim 8, wherein said second program code means further comprises program code means for causing the processor to generate at least one managed object of said second managed object tree in response to said step of identifying.

10. The computer program product of claim 8, wherein said second program code means further comprises program code means for causing the processor to determine the type of said topology transformation rule.

11. In a network management system having a network manager and at least one network element, wherein the network manager has a plurality of managed object trees, a system for software versioning, comprising:

means for identifying a first managed object tree having at least one managed object to be upgraded;

means for creating a second managed object tree having at least one managed object by transforming the topology of said first managed object tree; and

means for performing data transformation of said at least one managed object of said second managed object tree.

12. The system of claim 11, wherein said means for creating further comprises means for traversing said first managed object tree and building a list of said at least one managed object of said first managed object tree.

13. The system of claim 11, wherein said means for creating further comprises means for identifying a topology transformation rule associated with said at least one managed object of said first managed object tree.

14. The system of claim 13, wherein said means for creating further comprises means for generating at least one managed object of said second managed object tree in response to said step of identifying.

15. The system of claim 13, wherein said means for creating further comprises means for determining the type of said topology transformation rule.

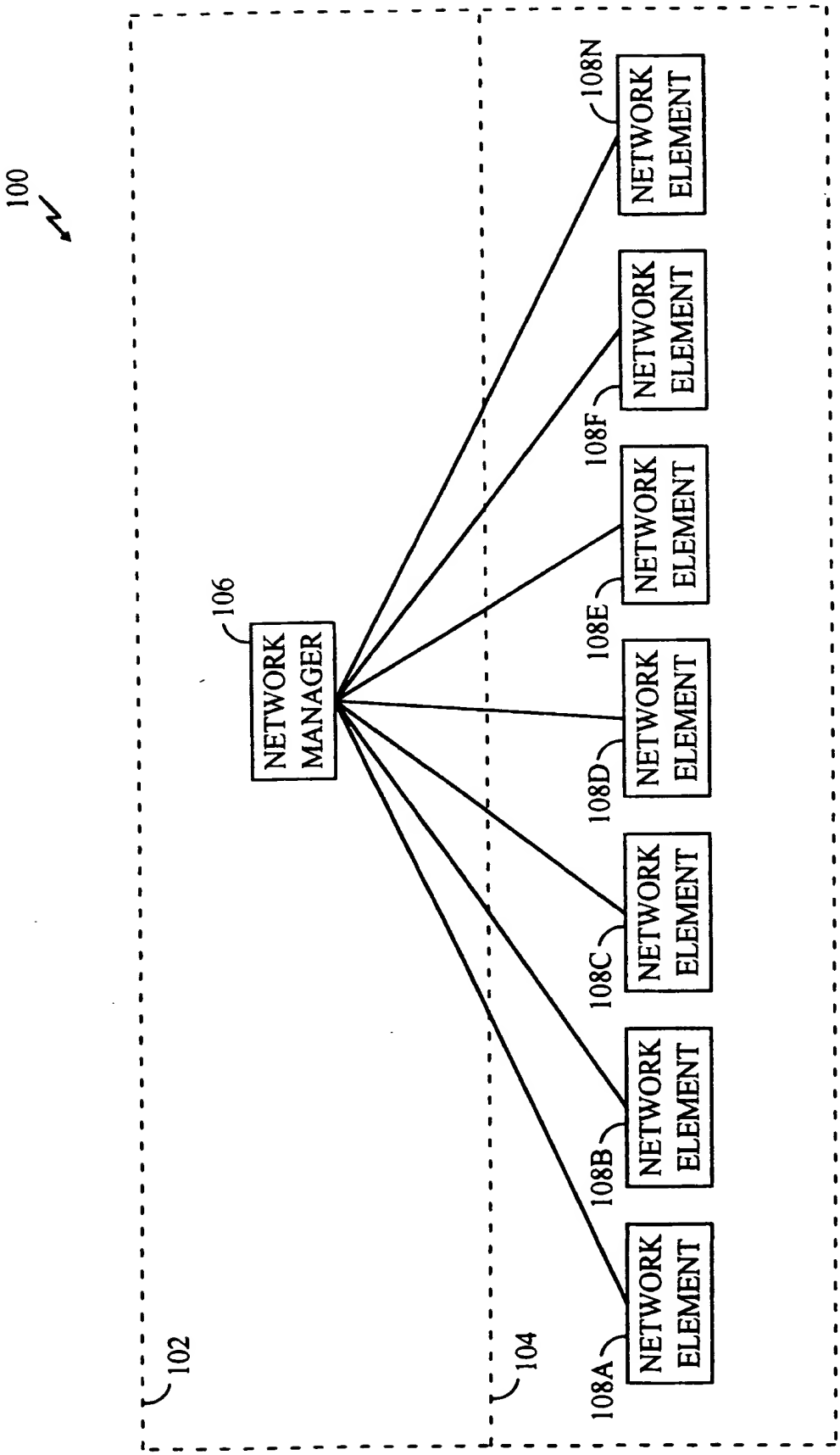

100
102
104
106
NETWORK MANAGER
108A
NETWORK ELEMENT
108B
NETWORK ELEMENT
108C
NETWORK ELEMENT
108D
NETWORK ELEMENT
108E
NETWORK ELEMENT
108F
NETWORK ELEMENT
108N
NETWORK ELEMENT

FIG. 1

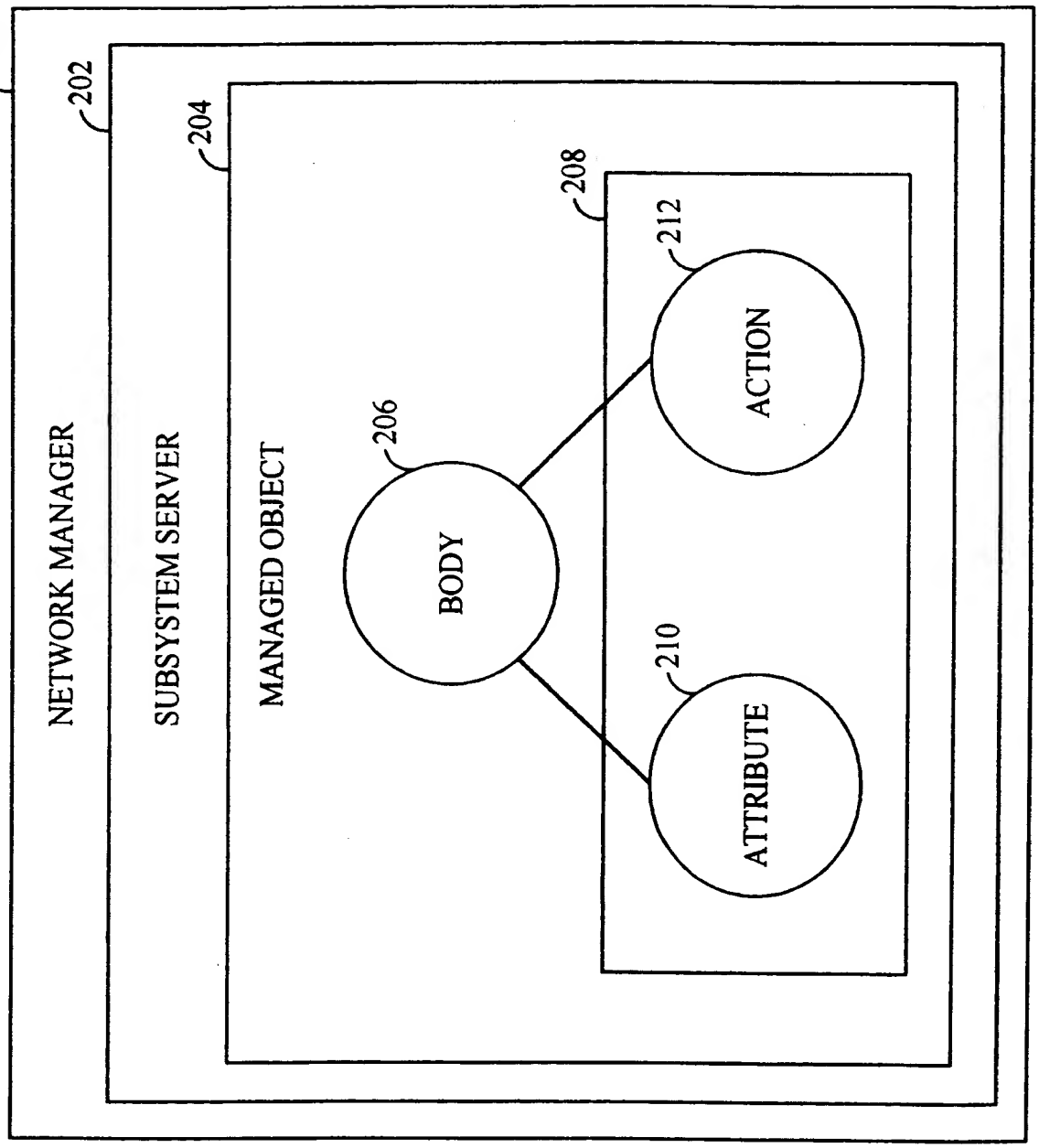


FIG. 2

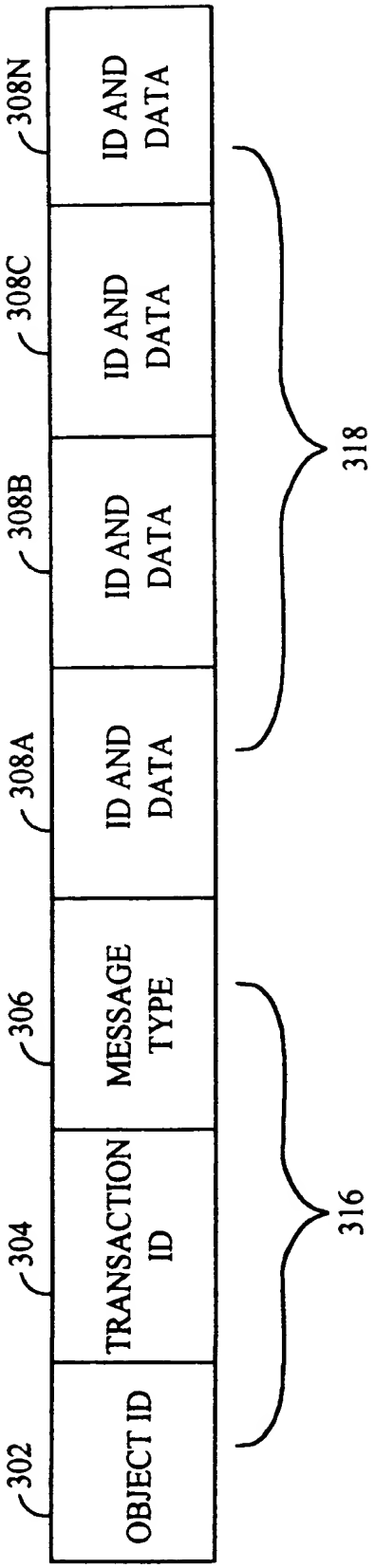

302
OBJECT ID
304
TRANSACTION ID
306
MESSAGE TYPE
308A
ID AND DATA
308B
ID AND DATA
308C
ID AND DATA
308N
ID AND DATA
316
318

FIG. 3

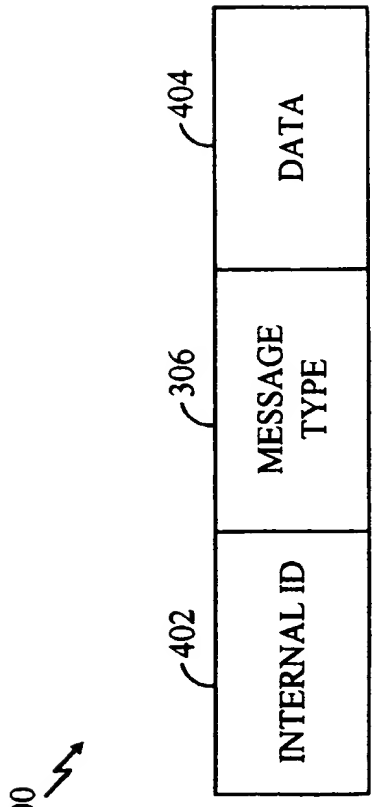

400
402
INTERNAL ID
306
MESSAGE TYPE
404
DATA

FIG. 4

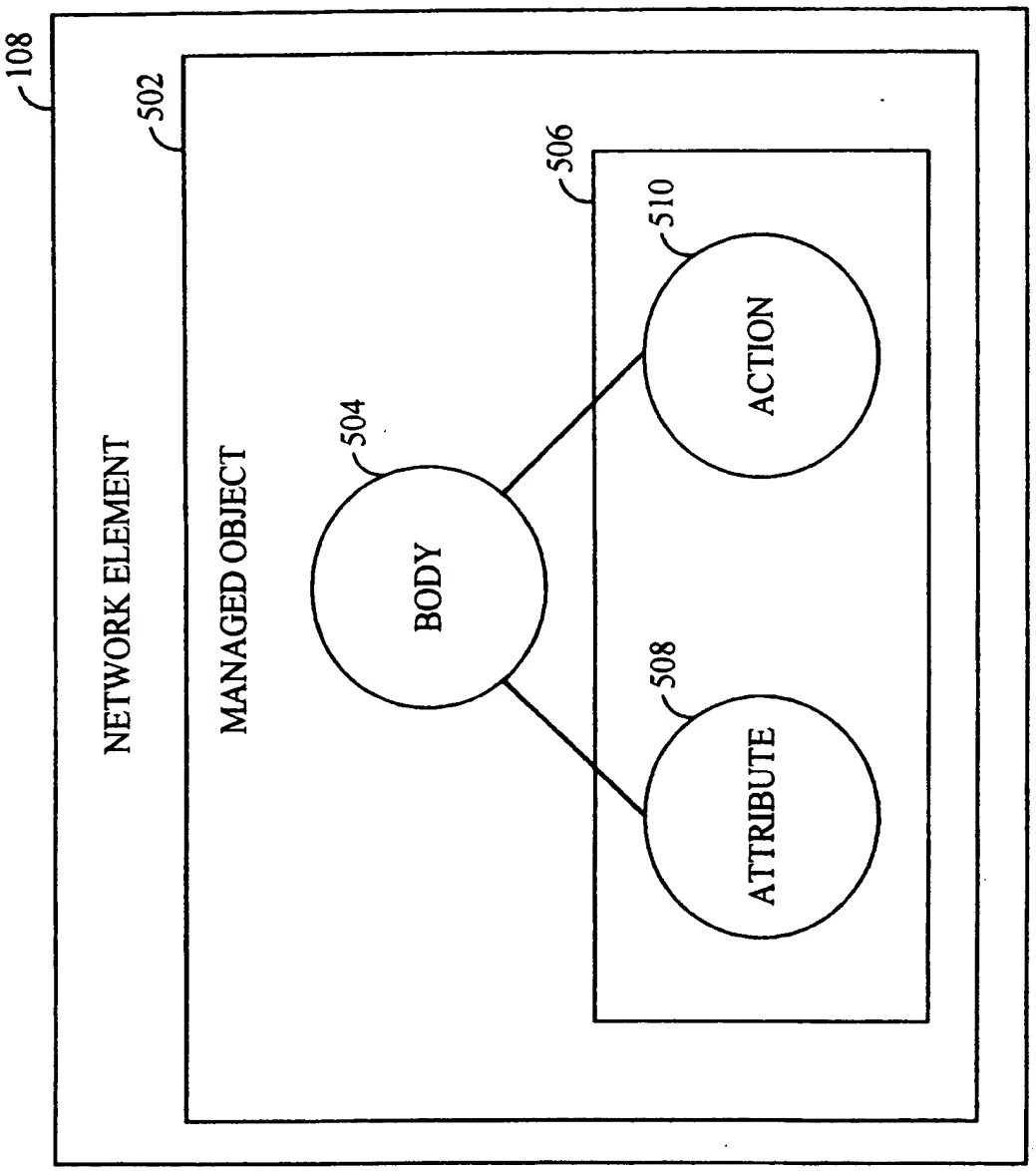

108
NETWORK ELEMENT
502
MANAGED OBJECT
504
BODY
506
510
ACTION
508
ATTRIBUTE

FIG. 5

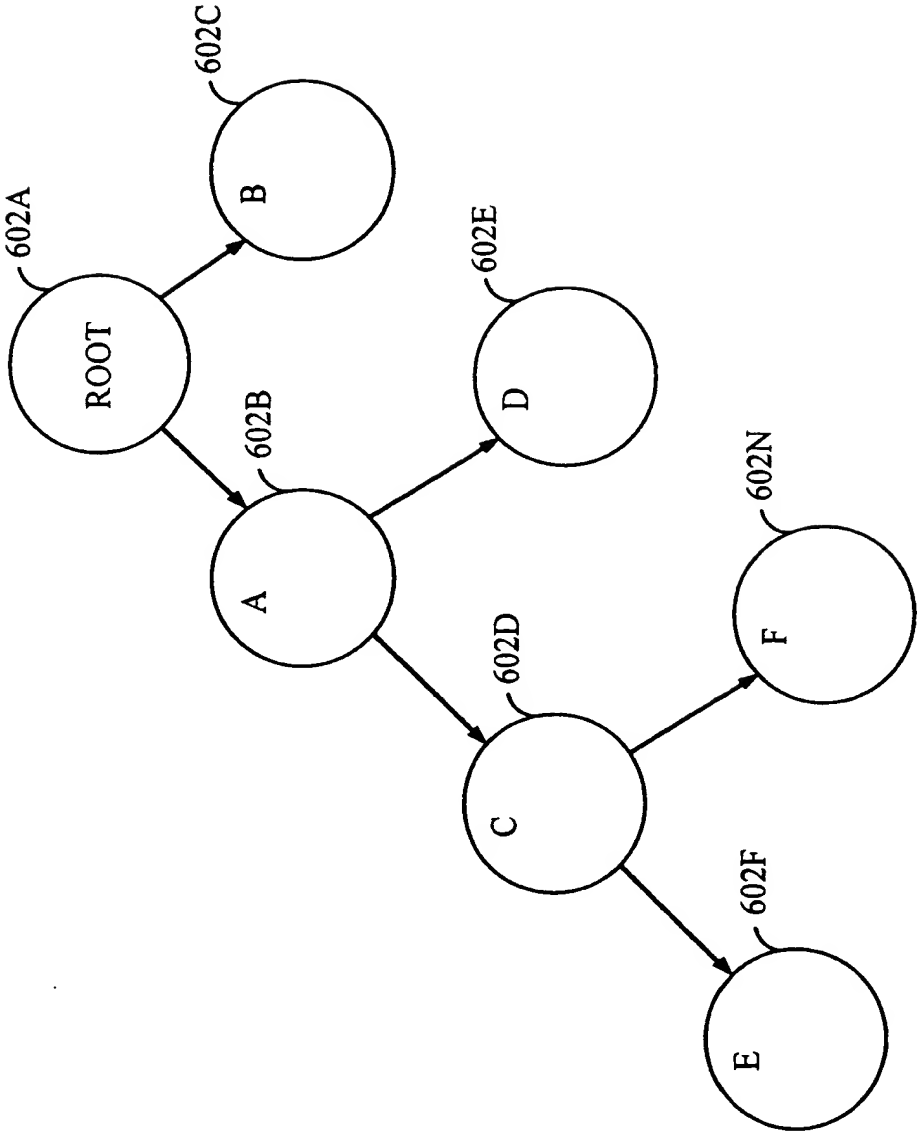


FIG. 6

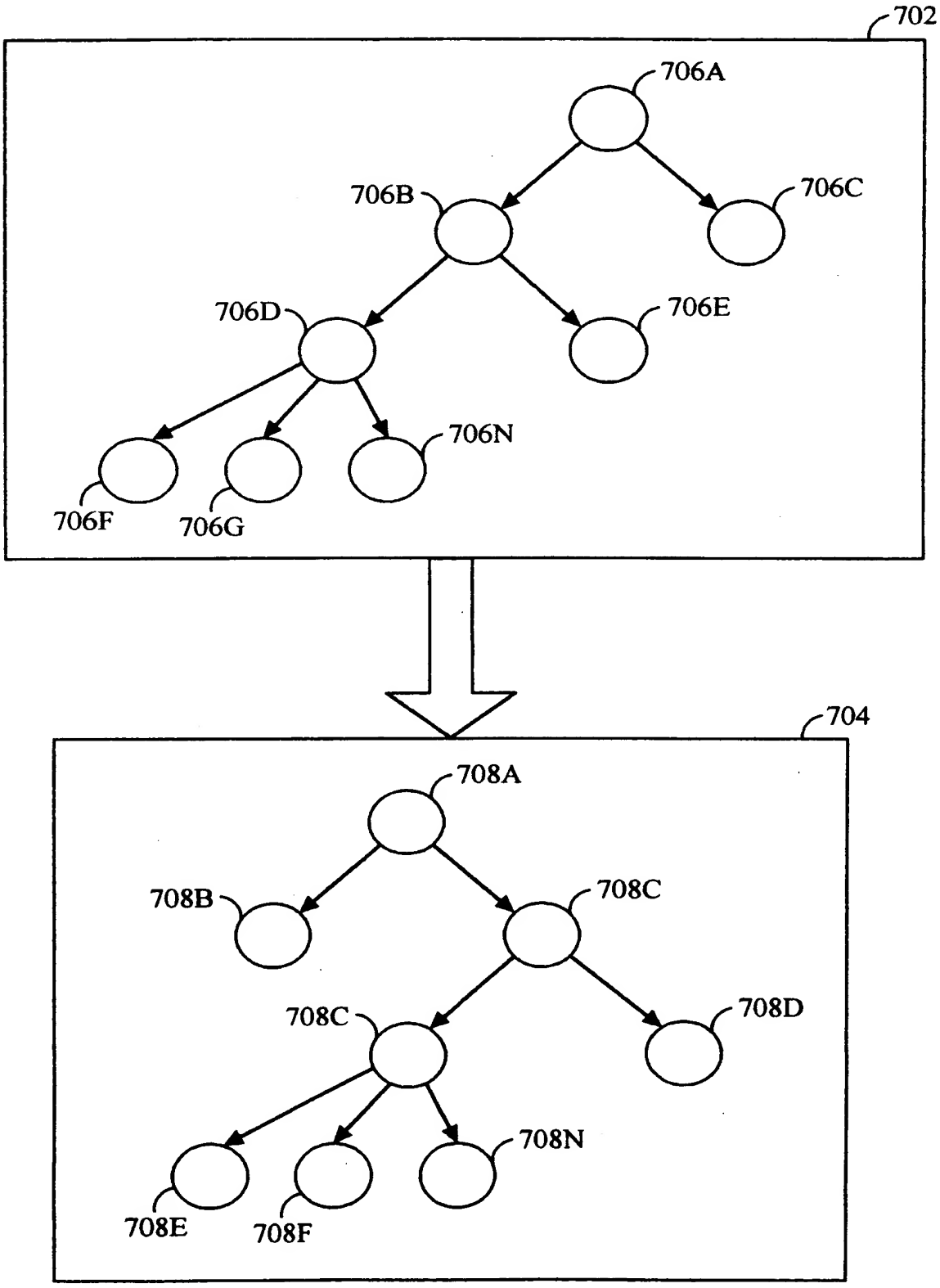

702
706A
706B
706C
706D
706E
706N
706F
706G
704
708A
708B
708C
708C
708D
708N
708E
708F

FIG. 7

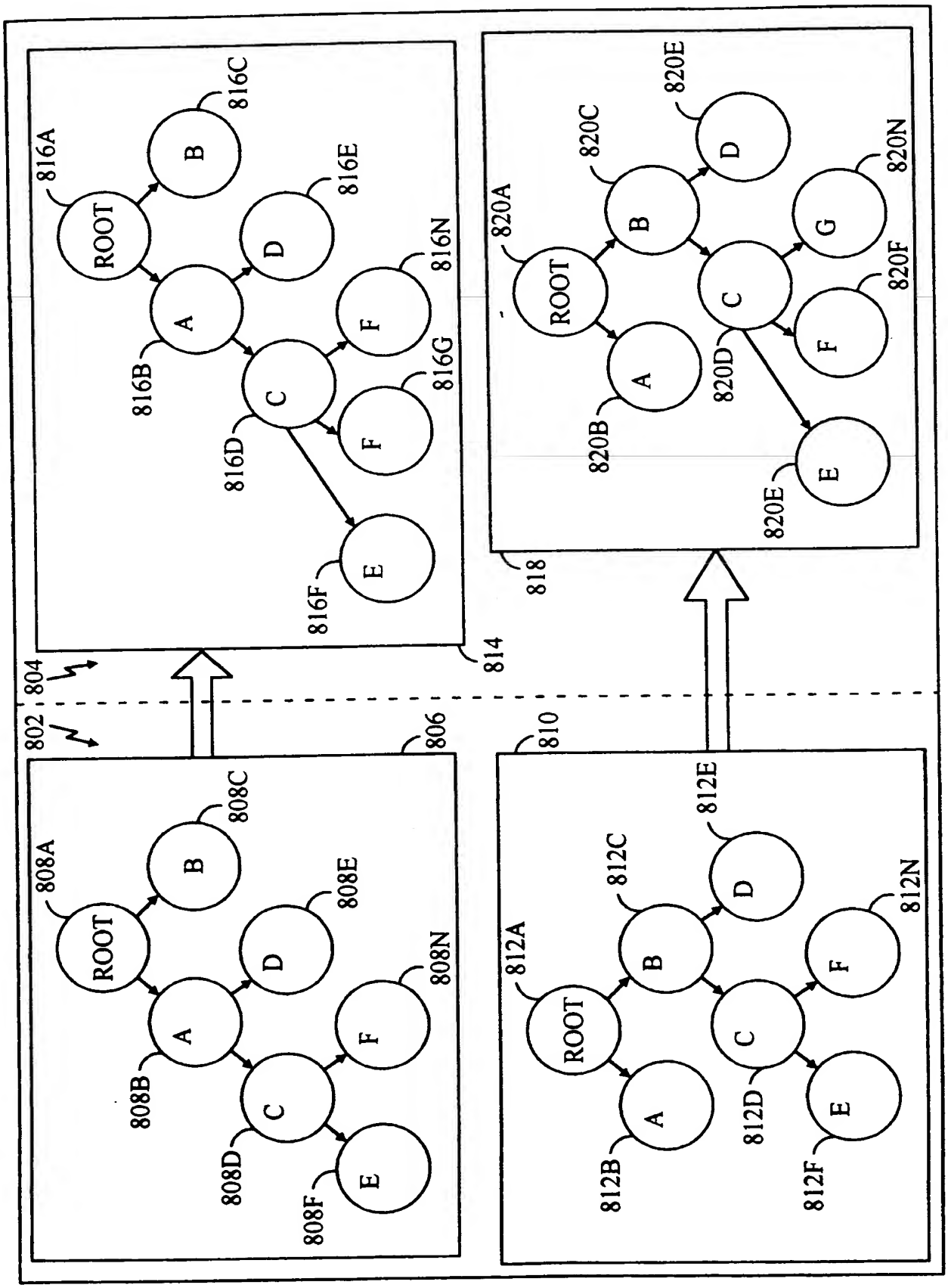

802
804
806
808A
ROOT
808B
A
808C
B
808D
C
808E
D
808F
E
808N
F
810
812A
ROOT
812B
A
812C
B
812D
C
812E
D
812F
E
812N
F
814
816A
ROOT
816B
A
816C
B
816D
C
816E
D
816F
E
816G
F
816N
F
818
820A
ROOT
820B
A
820C
B
820D
C
820E
D
820E
E
820F
F
820N
G

FIG. 8

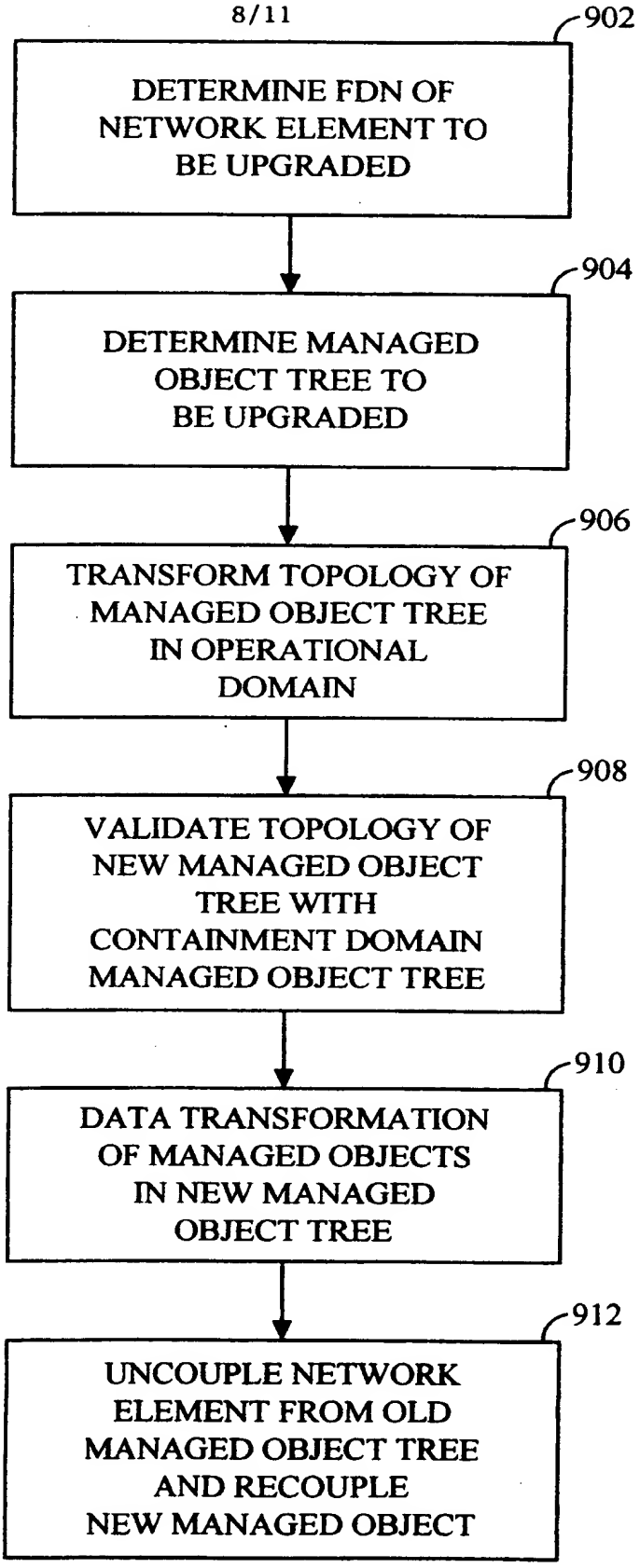


FIG. 9

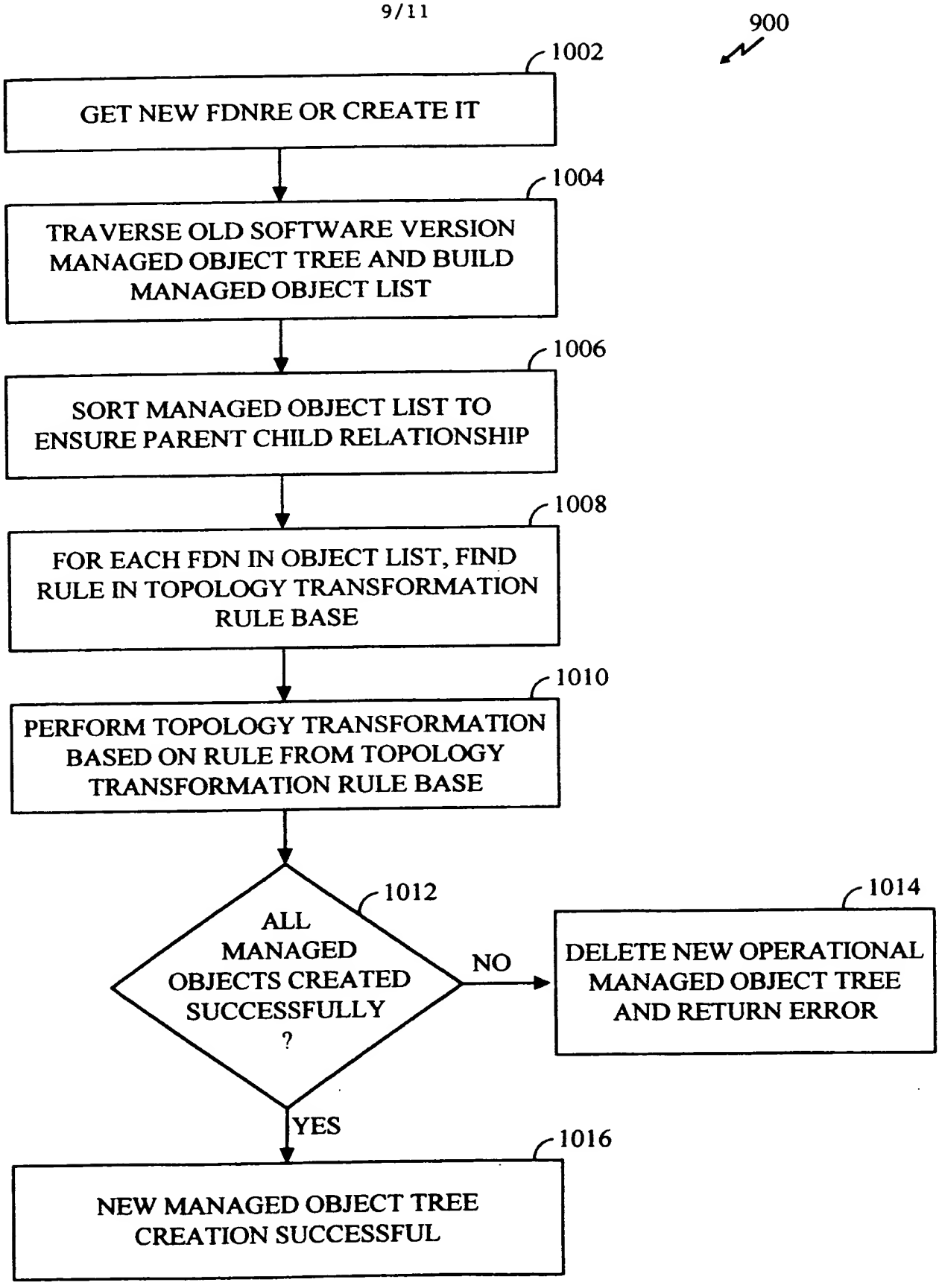

900
1002
GET NEW FDNRE OR CREATE IT
1004
TRAVERSE OLD SOFTWARE VERSION MANAGED OBJECT TREE AND BUILD MANAGED OBJECT LIST
1006
SORT MANAGED OBJECT LIST TO ENSURE PARENT CHILD RELATIONSHIP
1008
FOR EACH FDN IN OBJECT LIST, FIND RULE IN TOPOLOGY TRANSFORMATION RULE BASE
1010
PERFORM TOPOLOGY TRANSFORMATION BASED ON RULE FROM TOPOLOGY TRANSFORMATION RULE BASE
1012
ALL MANAGED OBJECTS CREATED SUCCESSFULLY ?
NO
1014
DELETE NEW OPERATIONAL MANAGED OBJECT TREE AND RETURN ERROR
YES
1016
NEW MANAGED OBJECT TREE CREATION SUCCESSFUL

FIG. 10

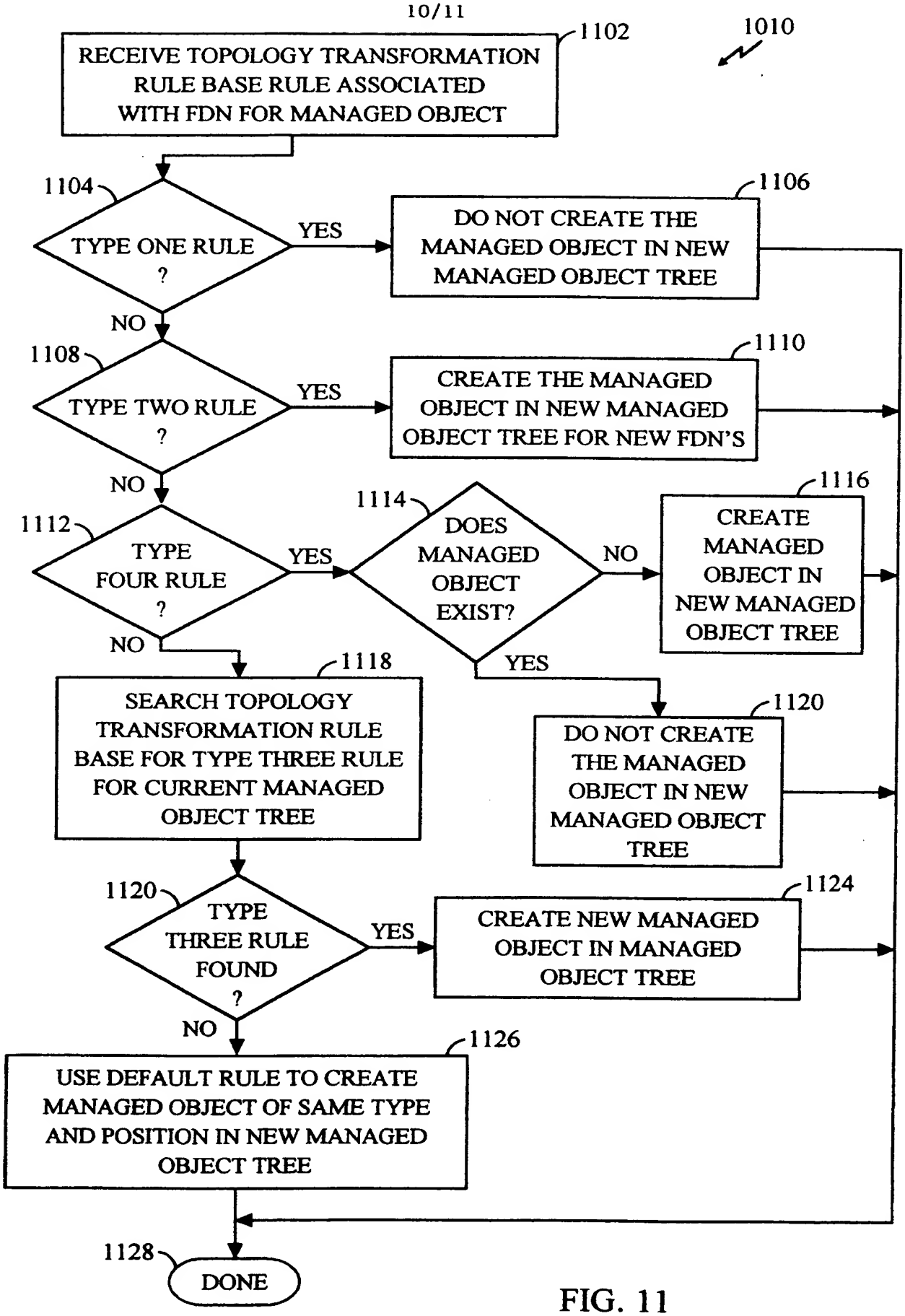


FIG. 11

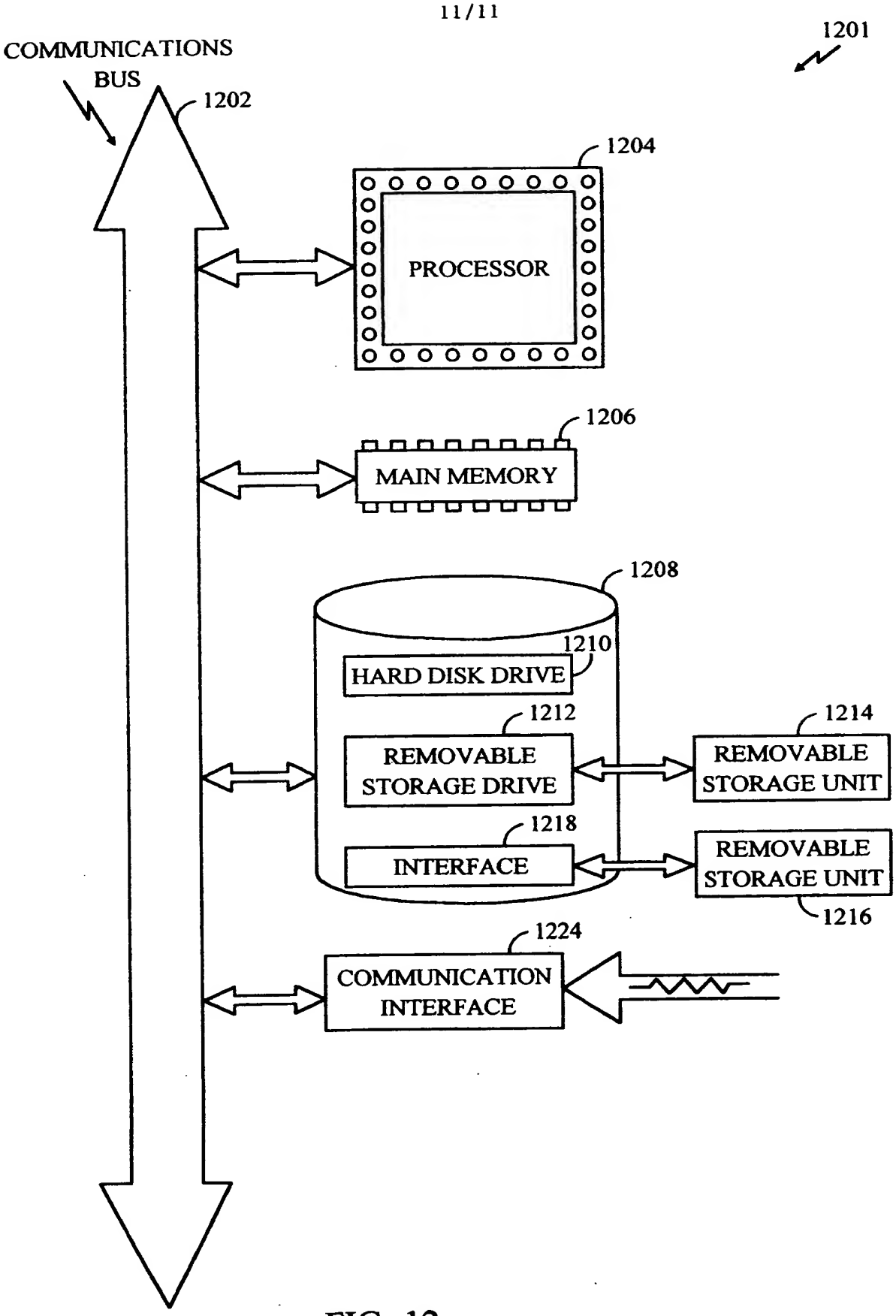

1201
COMMUNICATIONS BUS
1202
1204
PROCESSOR
1206
MAIN MEMORY
1208
1210
HARD DISK DRIVE
1212
REMOVABLE STORAGE DRIVE
1214
REMOVABLE STORAGE UNIT
1218
INTERFACE
REMOVABLE STORAGE UNIT
1216
1224
COMMUNICATION INTERFACE

FIG. 12

INTERNATIONAL SEARCH REPORT

Int tional Application No
PCT/US 98/27390

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L12/24

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	START K., PATEL A.: "The Distribution Management of srevice software" COMPUTER STANDARDS AND INTERFACES, vol. 17, no. 3, 1 June 1995, pages 291-301, XP004008584 see the whole document ---	1-3,6-8, 11-13
A	WO 96 38951 A (DCS COMMUNICATION CORPORATION) 5 December 1996 see abstract see figure 19 see page 1, line 1 - page 3, line 31 see page 35, line 22 - page 42, line 10 --- -/--	1-3,6-8, 11-13

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
7 May 1999	27/05/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Cichra, M

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Int .tional Application No
PCT/US 98/27390

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	"Stratey for collecting software inventory information across a local area network" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 12, 1 December 1994, pages 275-276, XP002101960 see the whole document ---	1-3,6-8, 11-13
A	US 4 847 830 A (NETWORK EQUIPMENT TECHNOLOGIES) 11 July 1989 see the whole document -----	1-3,6-8, 11-13

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int tional Application No
PCT/US 98/27390

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9638951 A	05-12-1996	GB 2301717 A	11-12-1996
		AU 6034896 A	18-12-1996
		AU 6037796 A	18-12-1996
		AU 6149096 A	18-12-1996
		CN 1192831 A	09-09-1998
		EP 0872026 A	21-10-1998
		EP 0830775 A	25-03-1998
		WO 9638930 A	05-12-1996
		WO 9638967 A	05-12-1996
		US 5761429 A	02-06-1998
US 4847830 A	11-07-1989	AT 120919 T	15-04-1995
		AU 2824089 A	05-07-1989
		CA 1307350 A	08-09-1992
		DE 3853539 D	11-05-1995
		DE 3853539 T	14-12-1995
		EP 0396589 A	14-11-1990
		JP 2834505 B	09-12-1998
		JP 3502742 T	20-06-1991
		WO 8905551 A	15-06-1989